Indian Streams Research Journal
Vol - I , ISSUE - V [June 2011] **ISSN:-2230-7850**

A NOVEL DATA MINING TECHNIQUE TO AUTOMATE SOFTWARE TESTING

Seema Phogat[1] and Prof. Sumit Gill[2]
[1]Scholar, Singhania University.

INTRODUCTION

Usually the size of test suite becomes very large so with the constraints that the testers have it become difficult for them to perform complete testing[1] In such a situation, the tester has no choice but to run fewer test cases to stay within the allowed time and resource constraints. The problem for the tester is then to decide which test cases are the most important and should therefore be run. Here the test suite reduction comes into the picture and automation of testing comes into the picture which is the basis of this paper. In this paper a novel method is presented to automate and prioritize the test suite.

KEYWORDS
Optimal testing, Heuristic Testing, ACTACMIN, APFD, Regression Testing.

LITERATURE SURVEY

Previous test suite minimization research has involved two main categories of minimization techniques [2, 3, 4]:

1. *Optimal techniques*, which attempt to compute optimally-minimized suites at the cost of potentially high runtime; and
2. *Heuristics*, which attempt to find near-optimal solutions more quickly.

The following paragraphs discuss the work related to test suite minimization
1. ATACMIN **–** A tool called ATACMIN, which is part of the ATAC tool package [2], contains an implementation that computes optimally-minimized suites, given a set of coverage requirements and the set of test cases satisfying each requirement. The approach taken by the tool is to implicitly enumerate subsets until an optimal minimized suite is found. While this algorithm clearly has theoretical runtime exponential in the worst case, in practice the execution is quite fast with relatively small suites. The group of researchers Wong et al. has conducted a series of empirical studies into test suite minimization using ATACMIN's optimally minimized suites.
2. Additional Fault Detection Effectiveness Research – The majority of research involving the notion of fault detection effectiveness has often employed this notion as a means for comparison between two or more concepts or techniques. For example, the minimization research that has already been described often used the notion of fault detection effectiveness to compare different minimization techniques. Similarly, fault detection effectiveness has often been used outside the realm of test suite minimization in order to compare things such as coverage criteria. I will now discuss work related to fault detection effectiveness that is not applied to the area of test suite minimization.[4,5].

MOTIVATION

The idea to reduce the test suite got its motivation from the following observations:

- Test suite grows exponentially and executing all these test cases is not possible. Therefore, a method is required to reduce them is required.
- Existing test suite minimization techniques attempt to remove the test cases that are redundant with respect to coverage criteria. However, removing the test cases in such a way significantly reduces the fault detection capability of the test suite.
- Non-prioritized test suite does not give an efficient Average Percentage of Fault Detection (APFD) value.
- Regression testing is considered to be the biggest hurdle in software testing as a small change in program leads to the creation of the test suite again.
- The du-paths which are non-dc are supposed to be more problematic.
- While testing the modified program with regression testing, the prioritized test suite for original program does not work. This happens as after modification, the new du-paths may get introduced and some of these du-paths may also be non-dc; meaning these would be more prone to errors.

OUR ATTEMPT

When a minimization algorithm selects the next test case to add to the reduced suite according to the primary minimization criterion, the other tests needs to be identified, given that the test case just selected by the minimization algorithm have become redundant with respect to the primary criterion. Among those redundant tests, it can be checked whether or not each test is also redundant with respect to the secondary criterion. If a test is redundant, it is thrown away. On the contrary, if a test is not redundant with respect to the secondary criterion, then the test case is added to the reduced suite. The original minimization algorithm is then to continue and select the next test according to the primary coverage criterion. Once the reduced test suite is prepared, it needs to be prioritized. In order to do so, the test suite is subjected to prioritization techniques.

PROPOSED ALGORITHM

Input:

- Minimization technique
- Test case set
- First and second coverage criteria.
- Prioritization technique

Output:

- Reduced test suite

Steps:

- Initially our reduced test suite (RTS) is empty and other data structure is initialized
- Run the existing minimizing algorithm on original test suite.
- It lists the test cases, which are not redundant according to primary criteria. Then add these test cases to the RTS and remove the added test cases from the original test suite.
- Now for those test cases, which are redundant according to first criteria, are subjected to test the redundancy according to second criteria.
- Test cases, which are redundant according to first criteria but not redundant according to second criteria, are also added RTS and remove the added test cases from the test suite.
- Remove those test cases, which are redundant according to first and second criteria from the test suite.

- The above-mentioned steps are followed until our test suite become empty.
- Last step involves prioritizing the test cases so that better fault detection capability can be achieved.

EXPLANATION

Step 1 (Initialization) - This step involves initializing the data structures and the reduced test suite that will be used in the algorithm later.
Step 2 (Select the New Test Case According to the Primary Criterion) - The existing minimization algorithm selects the next test case to include in the reduced suite according to the primary coverage requirements. This test is added to the reduced suite and is removed from the candidate set of test cases [1].
Step 3 (Identify Redundant Tests with Respect to the Primary Criterion) - Next, given the test case just selected according to the primary criterion, other test cases are identified in the candidate set of tests that have become redundant with respect to the reduced suite according to the primary criterion. These redundant tests are removed from the candidate set of tests since they will never be selected in the future by the existing minimization algorithm (according to the primary requirement set).
***Step 4 (Check the Redundant Sets Against Secondary Coverage Criteria*) -** This is the most important part of our approach, where selective coverage redundancy is considered. In this step, the set of primary coverage-redundant tests are analyzed. As long as a test case exists in this redundant set contributing to the secondary requirement coverage of the reduced suite, the next test case is selected that contributes most to the secondary requirement coverage in the reduced suite. After all redundant tests have been processed (some may be selected and some may not be selected), the redundant set is emptied and the existing minimization algorithm is allowed to continue selecting the next test case according to the primary coverage criterion.
Step 5 (Prioritizing to Get Better Testing Quality & Results) - Data-flow testing [2] monitors the lifecycle of a piece of data and looks out for inappropriate usage of data during definition, use in predicates, computations and termination (killing). It identifies potential bugs by examining the patterns in which that piece of data is used. While identifying the test cases of data flow testing, there may be large number of test cases. It may not be possible for a tester to execute all the test cases identified in this huge test suite due to time and cost constraints. Therefore, there is a need to prioritize the test cases so that the important ones are executed first that identify the critical bugs earlier. The du-paths which are not dc-paths (non-dc) are supposed to be more problematic from viewpoint of a tester. It means that there may be more bugs in the du-paths, which are not dc-paths. So the test cases based on this concept have been prioritized in this work for original programs. While testing modified programs in regression testing, the prioritized test-suite for original program will not work because after modification new du-paths may get introduced and some of these du-paths may also be non-dc i.e. these paths are more prone for errors. Such newly introduced non-dc paths should be given more priority over other test cases. It may happen that due to modification in the program some existing dc paths become non-dc. The test cases corresponding to such paths are filtered given next priority.Even after applying these methods for prioritizing test cases for regression, there may be a big set of test cases corresponding to dc paths. Taking all such test cases at the same priority may make testing process less effective. So control-structure weighted regression test case prioritization technique is applied to make the testing and prioritization process more effective.

CONCLUSION AND FUTURE WORK

Data flow testing identifies potential bugs by examining the patterns in which that piece of data is used. But the problem with this kind of testing is that there are large numbers of test cases, which increase the size of test suite. It may not be possible to execute all the test cases as it increases the time, effort and cost of the project. Therefore, there is a need to prioritize the test cases in data flow testing. A method has been referred in this work in this direction. The technique that has been researched upon for test case prioritization of data flow test cases is based on non-dc paths in the program. It is a beneficial technique for the purpose of saving resources such as time and cost. It proved to be effective because it identifies and executes the important test cases first and help in reducing the time, effort, and cost of testing. To analyze the above method, a set of three programs have been taken and to validate the prioritized test suite, the APFD (Average Percentage of Fault Detection) metric has been taken. Then a technique for regression test suite prioritization is taken. This technique is based on the newly introduced non-dc paths and paths which have changed into non-dc from dc paths in the program, due to modification of original program. However, in the above technique, no consideration was given to the complexity of the structure and statements where the changes have occurred. So, a new technique called control-structure weighted test case prioritization is taken into consideration the complexity of the statements and structure of program where the change has occurred. Then taking a program and making a prioritized test suite for it show the procedure of applying this technique. This method proves helpful in finding bugs early, thereby reducing the time, effort and cost of the project. In the method for prioritizing the test cases corresponding to du paths, a weight has been fixed for various programming constructs. But the values fixed can be more refined by experimenting the method for large and more complex programs. In this method the control structure and programming constructs weights are used for only du-dc paths, but in future the same method can be applied to dc paths.

REFERENCES

1. H. Agrawal. “Dominators, Super Blocks, and Program Coverage.” 21st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. 25-34, Portland, Oregon, January 1994.
2. H. Agrawal. “Efficient Coverage Testing Using Global Dominator Graphs.” Proceedings of the 1999 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering. Toulouse, France, 1999.
3. J. H. Andrews, L. C. Briand, and Y. Labiche “Is Mutation an Appropriate Tool for Testing Experiments?” 27th International Conference on Software Engineering. 402-411, St. Louis, Missouri, 2005.
4. M. Balcer, W. Hasling, and T. Ostrand. “Automatic Generation of Test Scripts from Formal Test Specifications.” Proc. of the 3rd Symp. on Software Testing, Analysis, and Verification. 210-218, Key West, Florida, December 1989.
5. J. Black, E. Melachrinoudis, and D. Kaeli. “Bi-Criteria Models for All-Uses Test Suite Reduction.” 26th Int’l Conf. on Software Engineering. Edinburgh, Scotland, May 2004.
6. T. Y. Chen and M. F. Lau. “Dividing Strategies for the Optimization of a Test Suite.” Information Processing Letters. 60(3):135-141, March 1996.
7. T. Y. Chen and M. F. Lau. “Heuristics Towards the Optimization of the Size of a Test Suite.” Proc. 3rd Int’l Conf. on Software Quality Management. Vol. 2, 415-424, Seville, Spain, April 1995.
8. V. Chvatal. “A Greedy Heuristic for the Set-Covering Problem.” Mathematics of Operations Research. 4(3), August 1979.
9. R. A. DeMillo and A. P. Mathur. “On the Use of Software Artifacts to Evaluate the Effectiveness of Mutation Analysis for Detecting Errors in Production Software.” Technical Report SERC-TR-92-P, Software Engineering Research Center, Purdue University, West Lafayette, IN, August 19, 1994.